



# TensorFlow-ZenDNN User Guide

Publication # **57301**

Revision # **4.0**

Issue Date **January 2023**

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

---

## **Trademarks**

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Dolby is a trademark of Dolby Laboratories.

ENERGY STAR is a registered trademark of the U.S. Environmental Protection Agency.

HDMI is a trademark of HDMI Licensing, LLC.

HyperTransport is a licensed trademark of the HyperTransport Technology Consortium.

Microsoft, Windows, Windows Vista, and DirectX are registered trademarks of Microsoft Corporation.

MMX is a trademark of Intel Corporation.

OpenCL is a trademark of Apple Inc. used by permission by Khronos.

PCIe is a registered trademark of PCI-Special Interest Group (PCI-SIG).

Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

## **Dolby Laboratories, Inc.**

Manufactured under license from Dolby Laboratories.

## **Rovi Corporation**

This device is protected by U.S. patents and other intellectual property rights. The use of Rovi Corporation's copy protection technology in the device must be authorized by Rovi Corporation and is intended for home and other limited pay-per-view uses only, unless otherwise authorized in writing by Rovi Corporation.

Reverse engineering or disassembly is prohibited.

USE OF THIS PRODUCT IN ANY MANNER THAT COMPLIES WITH THE MPEG-2 STANDARD IS EXPRESSLY PROHIBITED WITHOUT A LICENSE UNDER APPLICABLE PATENTS IN THE MPEG-2 PATENT PORTFOLIO, WHICH LICENSE IS AVAILABLE FROM MPEG LA, L.L.C., 6312 S. FIDDLERS GREEN CIRCLE, SUITE 400E, GREENWOOD VILLAGE, COLORADO 80111.

# Contents

---

<b>Revision History</b>	<b>6</b>
<b>Chapter 1 Installing ZenDNN with TensorFlow</b>	<b>7</b>
1.1 Binary Release Setup	7
1.1.1 Conda	7
1.1.2 TensorFlow v2.10	8
<b>Chapter 2 Directory Structure</b>	<b>9</b>
<b>Chapter 3 High-level Overview</b>	<b>10</b>
<b>Chapter 4 TensorFlow CNN Benchmarks</b>	<b>11</b>
<b>Chapter 5 TensorFlow v2.10</b>	<b>14</b>
<b>Chapter 6 Environment Variables</b>	<b>15</b>
<b>Chapter 7 Tuning Guidelines</b>	<b>18</b>
7.1 System	18
7.2 Environment Variables	18
7.3 Thread Wait Policy	19
7.4 Thread Affinity	19
7.5 Non-uniform Memory Access	20
7.5.1 numactl	20
7.5.2 Concurrent Execution	21
7.6 Transparent Huge Pages	21
7.7 Batch Size	22
7.8 Memory Allocators	22
7.8.1 TCMalloc	22
7.8.2 JEMalloc	22
7.8.3 Usage	23
<b>Chapter 8 Convolution Algo Logic</b>	<b>24</b>
<b>Chapter 9 Limited Precision Support</b>	<b>25</b>
<b>Chapter 10 License</b>	<b>26</b>
<b>Chapter 11 Technical Support</b>	<b>27</b>

## List of Figures

---

Figure 1.	ZenDNN Library. ....	10
-----------	----------------------	----

## List of Tables

---

Table 1.	ZenDNN Environment Variables-Generic . . . . .	15
Table 2.	ZenDNN Environment Variables-Optimization . . . . .	16
Table 3.	System Specification . . . . .	18
Table 4.	Convolution Algo Logic . . . . .	24

## Revision History

---

Date	Revision	Description
January 2023	4.0	<ul style="list-style-type: none"><li>• Updated supported TensorFlow versions.</li><li>• Added section 1.1.1.</li><li>• Added sections 7.3 through 7.8.</li><li>• Updated chapters 8 and 9.</li></ul>
June 2022	3.3	Updated supported TensorFlow versions.
December 2021	3.2	<ul style="list-style-type: none"><li>• Updated the supported TensorFlow version.</li><li>• Removed the obsolete environment variables.</li></ul>
August 2021	3.1	Updated supported TensorFlow versions.
April 2021	3.0	Initial version.

# Chapter 1 Installing ZenDNN with TensorFlow

**Note:** Refer to the ZenDNN v4.0 User Guide before starting the installation.

In this release, ZenDNN library is supported for TensorFlow v2.10. This is a baseline release for TensorFlow v2.10 with:

- FP32 support
- Only AMD UIF INT8 model support
- Limited support for BF16 on AMD UIF ResNet50 and VGG16
- INT16 is not supported

## 1.1 Binary Release Setup

This section describes the procedure to setup the ZenDNN binary release for TensorFlow v2.10.

### 1.1.1 Conda

Complete the following steps to setup Conda:

1. Refer to Anaconda documentation (<https://docs.anaconda.com/anaconda/install/linux/>) to install Anaconda on your system.
2. Create and activate a Conda environment which will house all the TF-ZenDNN specific installations:

```
conda create -n tf-v2.10-zendnn-v4.0-rel-env python=3.8
conda activate tf-v2.10-zendnn-v4.0-rel-env
```

Ensure that you install the TF-ZenDNN package corresponding to the Python version with which you created the Conda environment.

**Note:** If there is any conda environment named `tf-2.10-zendnn-v4.0-rel-env` already present, delete the conda environment `tf-2.10-zendnn-v4.0-rel-env` (using command `conda remove --name tf-2.10-zendnn-v4.0-rel-env --all`) before running scripts/`TF_ZenDNN_setup_release.sh`.

3. It is recommended to use the naming convention:

```
tf-v2.10-zendnn-v4.0-rel-env
```

#### 4. Install all the necessary dependencies:

```
pip install --upgrade pyparsing
pip install --upgrade appdirs
pip install --upgrade --no-deps --force-reinstall --no-cache-dir numpy==1.23.2 absl-py
pip install -U pip six wheel importlib-metadata setuptools mock future
pip install -U keras_applications --no-deps
pip install -U keras_preprocessing --no-deps
```

**Note:** For binary packages built with Python v3.7, it is recommended to use numpy v1.21.6 (numpy==1.21.6).

### 1.1.2 TensorFlow v2.10

Complete the following steps to install the ZenDNN binary release:

1. Copy the zipped release package to the local system being used. The name of the release package will be similar to *TF\_v2.10\_ZenDNN\_v4.0\_Python\_v3.8.zip*.
2. Execute the following commands:
  - a. `unzip TF_v2.10_ZenDNN_v4.0_Python_v3.8.zip`
  - b. `cd TF_v2.10_ZenDNN_v4.0_Python_v*/`
  - c. `source scripts/TF_ZenDNN_setup_release.sh`

This installs the TensorFlow wheel package provided in the zip file.

**Note:** Ensure that it is sourced only from the folder *TF\_v2.10\_ZenDNN\_v4.0\_Python\_v\*/*.

The release binaries for TensorFlow v2.10 are now compiled with manylinux2014 and they provide compatibility with some older Linux distributions. The support for docker releases has been discontinued.

The Python release binaries are tested with the recent Linux distributions such as:

- Ubuntu 20.04 and later
- RHEL 9.0 and later

C++ Interface will work on operating systems (with glibc version 2.31 or later) such as:

- Ubuntu 20.04 and later
- RHEL 9.0 and later

## Chapter 2 Directory Structure

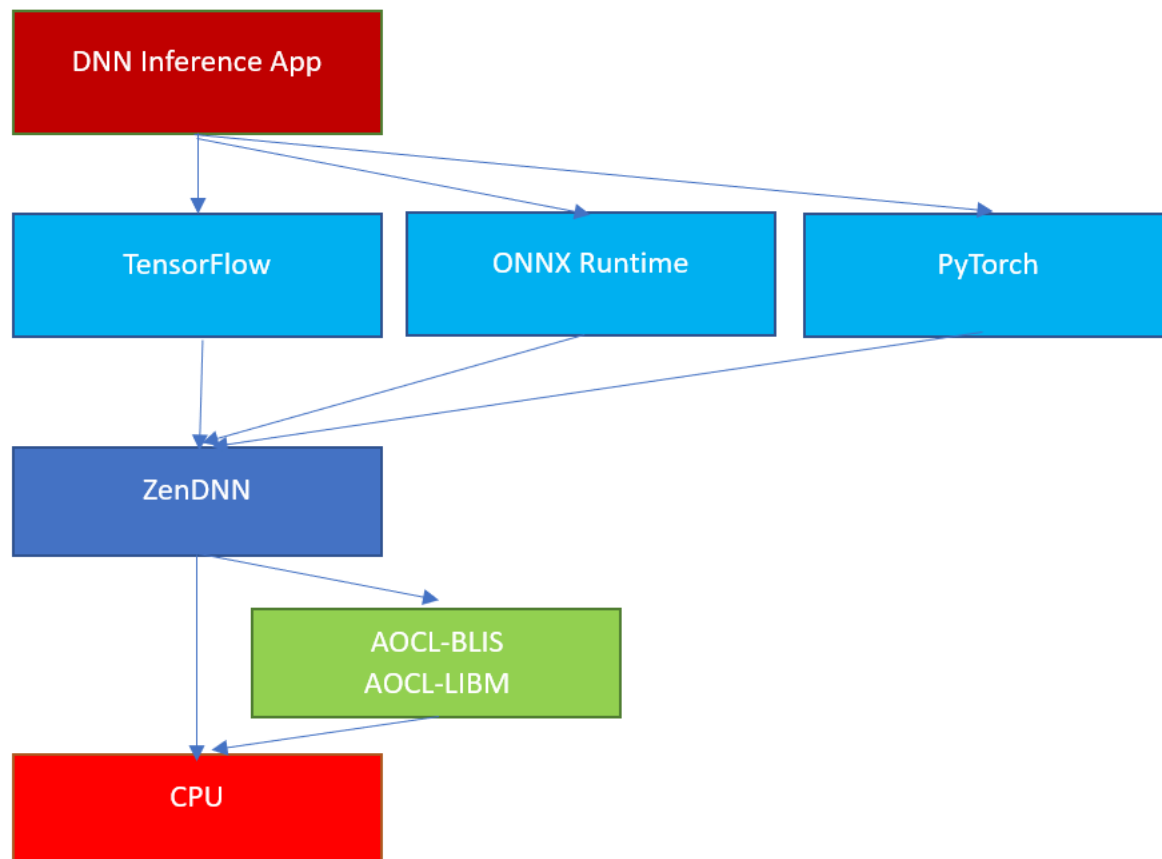
---

The release folder consists of a TensorFlow wheel (.whl), LICENSE and THIRD-PARTY-PROGRAMS files, and the following directory:

- *scripts* contains scripts to set up the environment and run benchmarks

## Chapter 3 High-level Overview

The following is a high-level block diagram for the ZenDNN library, which uses the AOCL-BLIS library internally:



**Figure 1. ZenDNN Library**

In the current release, ZenDNN is integrated with TensorFlow, PyTorch, and ONNX Runtime.

## Chapter 4 TensorFlow CNN Benchmarks

The benchmark scripts provide performance benchmarking at the TensorFlow level, printing latency and throughput results for AlexNet, GoogLeNet, InceptionV3, InceptionV4, ResNet50, ResNet152, VGG16, and VGG19 models.

Complete the following steps:

1. Download the TensorFlow CNN benchmarks repository from GitHub:

<https://github.com/tensorflow/benchmarks.git>

```
cd $ZENDNN_PARENT_FOLDER
git clone https://github.com/tensorflow/benchmarks.git $ZENDNN_PARENT_FOLDER/benchmarks
```

2. Export the environment variable `BENCHMARKS_GIT_ROOT` with the path to the benchmarks repository:

```
export BENCHMARKS_GIT_ROOT=$ZENDNN_PARENT_FOLDER/benchmarks
```

For latency, execute the following commands:

1. `cd TF_v2.10_ZenDNN_v4.0_Python_v*/`
2. `source scripts/zendnn_TF_env_setup.sh`
3. `source scripts/tf_cnn_benchmarks_latency.sh`

For throughput, execute the following commands:

1. `cd TF_v2.10_ZenDNN_v4.0_Python_v*/`
2. `source scripts/zendnn_TF_env_setup.sh`
3. `source scripts/tf_cnn_benchmarks_throughput.sh`

To run the individual models rather than the entire suite, execute the following commands:

```
cd $BENCHMARKS_GIT_ROOT/scripts/tf_cnn_benchmarks/
numactl --cpunodebind=<NPS> --interleave=<NPS> python tf_cnn_benchmarks.py --
device=cpu --model=<model_name> --data_format=NHWC --batch_size=$BATCH_SIZE --
num_batches=100 --num_inter_threads=1 --num_intra_threads=64 --nodistortions
```

Replace `<NPS>` with the following based on your number of NUMA nodes. Execute the command `lscpu` to identify the number of NUMA nodes for your machine:

- If you have 1 NUMA node, replace `<NPS>` with 0
- If you have 2 NUMA nodes, replace `<NPS>` with 0-1
- If you have 4 NUMA nodes, replace `<NPS>` with 0-3

Replace `<model_name>` with one of the following options:

- For AlexNet, replace <model\_name> with alexnet
- For GoogLeNet, replace <model\_name> with googlenet
- For InceptionV3, replace <model\_name> with inception3
- For InceptionV4, replace <model\_name> with inception4
- For ResNet50, replace <model\_name> with resnet50
- For ResNet152, replace <model\_name> with resnet152
- For VGG16, replace <model\_name> with vgg16
- For VGG19, replace <model\_name> with vgg19

While executing the commands, make a note of the following:

- For optimal settings, refer to the Tuning Guidelines section. Current setting refers to 96C, 2P, SMT=ON configuration.
- If the following warning is displayed on the terminal, it can be ignored. During the environment setup, there is an optional script to gather information about hardware, OS, Kernel, and BIOS and it requires a few utilities (lscpu, lstopo-no-graphics, dmidecode, and so on) to be present. If these utilities are not present, you may encounter the following warning(s):

```
scripts/gather_hw_os_kernel_bios_info.sh
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: lstopo-no-graphics: command not found
bash: _HW_LSTOPO_NUM_L2CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")
bash: _HW_LSTOPO_NUM_L3CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")
sudo: dmidecode: command not found
sudo: dmidecode: command not found
sudo: dmidecode: command not found
```

- If a warning similar to the following appears during benchmark runs, configure your GOMP\_CPU\_AFFINITY setting to match the number of CPU cores supported by your machine:

```
OMP: Warning #181: OMP_PROC_BIND: ignored because GOMP_CPU_AFFINITY is defined
```

```
OMP: Warning #123: Ignoring invalid OS proc ID 48
```

```
OMP: Warning #123: Ignoring invalid OS proc ID 49
```

```
.
```

```
.
```

```
.
```

```
OMP: Warning #123: Ignoring invalid OS proc ID 63
```

For example, if your CPU has 24 cores, your GOMP\_CPU\_AFFINITY should be set as "export GOMP\_CPU\_AFFINITY=0-23".

---

## Chapter 5 TensorFlow v2.10

---

In this release of ZenDNN:

- ZenDNN library is supported for TensorFlow v2.10.
- AMD Unified Inference Frontend (UIF) optimized models are supported. For the model details, refer to the AMD UIF documentation.
- TensorFlow v2.10 wheel file is compiled with GCC v9.3.1.
- TensorFlow v2.10 is expected to deliver similar or better performance as compared to TensorFlow v2.9.

## Chapter 6 Environment Variables

ZenDNN uses the following environment variables to setup paths and control logs:

**Table 1. ZenDNN Environment Variables-Generic**

Environment Variable	Default Value/User Defined Value
ZENDNN_LOG_OPTS	ALL: 0
ZENDNN_PARENT_FOLDER	Path to unzipped release folder
TF_ZEN_PRIMITIVE_REUSE_DISABLE	False
ZENDNN_ENABLE_MEMPOOL	The default value is set to 1, you can provide the value 0 to disable it. 1 is for Graph-based MEMPOOL and 2 is for Node-based MEMPOOL.
ZENDNN_PRIMITIVE_CACHE_CAPACITY	The default value is set to 1024, you can modify it as required <sup>a</sup> .
ZENDNN_TENSOR_BUF_MAXSIZE_ENABLE	0
OMP_DYNAMIC	FALSE
ZENDNN_INFERENCE_ONLY	Default value is set to 1. ZenDNN does not currently support training. You can set it to 0 when you want to enable vanilla training and inference.

The following is a list of environment variables to tune performance:

**Table 2. ZenDNN Environment Variables-Optimization**

Environment Variable	Default Value/User Defined Value
OMP_NUM_THREADS	The default value is set to 96. You can set it as per the number of cores in the user system <sup>a</sup> .
OMP_WAIT_POLICY	ACTIVE
OMP_PROC_BIND	FALSE
GOMP_CPU_AFFINITY	Set it as per the number of cores in the system being used. For example, use 0-95 for 96-core server.
ZENDNN_TENSOR_POOL_LIMIT	The default value is set to 32. For ZENDNN_ENABLE_MEMPOOL=2, you can modify to 256 for CNNs and 1024 for NLP models for optimal performance.
ZENDNN_INT8_SUPPORT	The default value is set to 0. You can modify it to 1 to enable the INT8 data type support. This works only with ZENDNN_CONV_ALGO=4. <i>Note: This environment variable is not required for AMD UIF models.</i>
ZENDNN_TF_CONV_ADD_FUSION_SAFE	The default value is set to 0. You can modify it to 1 to enable Conv, Add fusion. Currently it is safe to enable this switch for resnet50v1_5, resnet101, and inception_resnet_v2 models only.
ZENDNN_GEMM_ALGO	The default value is 3. You can modify it to one of the following: <ul style="list-style-type: none"> <li>• 0 = Auto</li> <li>• 1 = BLIS path</li> <li>• 2 = partial-BLIS</li> <li>• 3 = ZenDNN jit path</li> <li>• 4 = ZenDNN partial jit path</li> </ul> <i>Note: Auto is an experimental feature.</i>
ZENDNN_CONV_ALGO	Decides the convolution algorithm to be used in execution and the possible values are: <ul style="list-style-type: none"> <li>• 1 = im2row followed by GEMM</li> <li>• 2 = WinoGrad (fallback to im2row GEMM for unsupported input sizes)</li> <li>• 3 = Direct convolution with blocked inputs and filters</li> <li>• 4 = Direct convolution with blocked filters</li> </ul> The default value is set to 1.
ZENDNN_LOG_OPTS=FWK:4	Dump graph after ZenDNN rewrites pass for all the TensorFlow models.

**Table 2. ZenDNN Environment Variables-Optimization**

Environment Variable	Default Value/User Defined Value
TF_ENABLE_ZENDNN_OPTS	The default value is set to 1 and ZenDNN code path will be used. You can modify it to 0 to use native TensorFlow code path.

- a. You must set these environment variable explicitly.

**Note:** *There are a few other environment variables that are initialized by the setup script, however these are not applicable for the binary release setup.*

When source `scripts/zendnn_TF_env_setup.sh` is invoked, the script initializes all the environment variables except the one(s) which must be set manually. The environment variables **ZENDNN\_PARENT\_FOLDER** is initialized relative to the unzipped release folder. To ensure that the paths are initialized correctly, it is important that the script is invoked from the unzipped release folder.

## Chapter 7 Tuning Guidelines

The hardware configuration, OS, Kernel, and BIOS settings play an important role in performance. The details for the environment variables used on a 4<sup>th</sup> Gen AMD EPYC™ server to achieve the optimal performance numbers are as follows:

### 7.1 System

A system with the following specifications has been used:

**Table 3. System Specification**

<b>Model name</b>	4 <sup>th</sup> Gen AMD EPYC™ 9654P 96-Core Processor
<b>DPU MHz</b>	Up to 3.7 GHz
<b>No of Cores</b>	96
<b>1P/2P</b>	1
<b>SMT: Thread(s) per Core</b>	2
<b>Mem-Dims</b>	12x64 GB

**OS Used:** Ubuntu 20.04.02 LTS

### 7.2 Environment Variables

The following environment variables have been used:

```

ZENDNN_LOG_OPTS=ALL:0
TF_ENABLE_ONEDNN_OPTS=0
TF_ENABLE_ZENDNN_OPTS=1
OMP_NUM_THREADS=96
OMP_WAIT_POLICY=ACTIVE
OMP_PROC_BIND=FALSE
OMP_DYNAMIC=FALSE
ZENDNN_ENABLE_MEMPOOL=1
ZENDNN_GEMM_ALGO=3
ZENDNN_TENSOR_POOL_LIMIT=32
ZENDNN_TENSOR_BUF_MAXSIZE_ENABLE=0
ZENDNN_CONV_ALGO=1

```

```
ZENDNN_PARENT_FOLDER=/home/<user_id>/my_work
BENCHMARKS_GIT_ROOT=/home/<user_id>/my_work/benchmarks
ZENDNN_PRIMITIVE_CACHE_CAPACITY=1024
ZENDNN_INT8_SUPPORT=0
ZENDNN_INFERENCE_ONLY=1
ZENDNN_TF_CONV_ADD_FUSION_SAFE=0
```

Other than the ZENDNN environment variables, there are a few other parameters that influence the memory policy across the nodes, thread binding to the available physical cores. Considerable performance improvements may be achieved by setting these parameters carefully. Following sections describe the behavior and possible value for these parameters.

## 7.3 Thread Wait Policy

**OMP\_WAIT\_POLICY** environment variable provides options to the OpenMP runtime library based on the expected behavior of the waiting threads. It can take the abstract values **PASSIVE** and **ACTIVE**. The default value is **ACTIVE**. When **OMP\_WAIT\_POLICY** is set to **PASSIVE**, the waiting threads will be passive and will not consume the processor cycles. Whereas, setting it to **ACTIVE** will consume processor cycles.

*Note: For ZenDNN stack, setting **OMP\_WAIT\_POLICY** to **ACTIVE** may give better performance.*

## 7.4 Thread Affinity

To improve ZenDNN performance, the behavior of OpenMP thread can be guarded precisely with thread affinity settings. A thread affinity defined at start up cannot be modified or changed during runtime of the application. Following are the ways through which you can bind the requested OpenMP threads to the physical CPUs:

- **GOMP\_CPU\_AFFINITY** environment variable binds threads to the physical CPUs, for example:

```
export GOMP_CPU_AFFINITY="0 3 1-2 4-15:2"
```

This command will bind the:

- Initial thread to CPU 0
- Second thread to CPU 3
- Third and fourth threads to CPU 1 and CPU 2 respectively
- Fifth thread to CPU 4
- Sixth through tenth threads to CPUs 6, 8, 10, 12, and 14 respectively

Then, it will start the assigning back from the beginning of the list.

```
export GOMP_CPU_AFFINITY="0" binds all the threads to CPU 0.
```

- Kmp affinity belongs to LLVM OpenMP runtime library and is used by setting appropriate values for the environment variable **KMP\_AFFINITY**. It has the following syntax:

**KMP\_AFFINITY**=[<modifier>,...]<type>[,<permute>][,<offset>]

Example:

**export KMP\_AFFINITY='verbose,respect,granularity=fine,compact,1,0'** binds threads as close as possible to the master thread but on a different core. Once each core is assigned with one OpenMP thread, the remaining OpenMP threads are assigned in the same order as before, but on different thread context.

Among the two, **KMP\_AFFINITY** takes highest precedence followed by **GOMP\_CPU\_AFFINITY**. If none of the two is set, the host system will defer the assignment of threads to CPUs. Given the same thread binding (see example below), it is expected that both the affinity settings would give the same performance.

Example:

Following affinity settings should give the same thread bindings:

- **export GOMP\_CPU\_AFFINITY=0-95**
- **export KMP\_AFFINITY='verbose,respect,granularity=fine,compact,1,0'**

## 7.5 Non-uniform Memory Access

### 7.5.1 numactl

numactl provides options to run processes with specific scheduling and memory placement policy. It can restrict the memory binding and process scheduling to specific CPUs or NUMA nodes:

- **--cpunodebind=nodes**: Restricts the process to specific group of nodes.
- **--physcpubind=cpus**: Restricts the process to specific set of physical CPUs.
- **--membind=nodes**: Allocates the memory from the nodes listed. The allocation fails if there is not enough memory on the listed nodes.
- **--interleave=nodes**: Memory will be allocated in a round robin manner across the specified nodes. When the memory cannot be allocated on the current target node, it will fall back to the other nodes.

Example:

If <tensorflow\_script> is the application that needs to run on the server, then it can be triggered using numactl settings as follows:

```
numactl --cpunodebind=0-3 --membind=0-3 python <tensorflow_script>
```

The **interleave** option of numactl works only when the number nodes allocated for a particular application is more than one. **cpunodebind** and **physcpubind** behave the same way for ZenDNN stack, whereas **interleave** memory allocation performs better than **membind**.

## 7.5.2 Concurrent Execution

As every application, AI workload requires special considerations during performance tuning to get the best out of the non-uniform memory access (NUMA) enabled machine. Improvement in performance can be achieved by carefully analyzing memory access time, memory bandwidth, and congestion on the shared bus. These factors depend on how far away the allocated memory and the process that requested the memory are in the NUMA system. In NUMA machines, the local memory access is faster as compared to the remote memory access. Consider the following workload:

```
numactl --cpunodebind=0-3 --membind=0-3 python <tensorflow_script>
```

Performance can be optimized by partitioning the workload into multiple data shards and then running concurrently on more than one NUMA node. Following example shows the concurrent execution across 4 NUMA nodes:

```
numactl --cpunodebind=0 --membind=0 python <tensorflow_script> & numactl --cpunodebind=1 --membind=1 python <tensorflow_script> & numactl --cpunodebind=2 --membind=2 python <tensorflow_script> & numactl --cpunodebind=3 --membind=3 python <tensorflow_script>
```

The number of concurrent executions can be increased beyond 4 nodes. The following formula can be used to decide the number of concurrent executions to be triggered at a time:

```
Number Concurrent Executions = Number of Cores Per Socket / Numbers of Cores sharing L3 cache
```

This can also be extended to even cores. However, these details should be verified by the user empirically.

## 7.6 Transparent Huge Pages

Transparent Huge Pages (THPs) are a Linux kernel feature for memory management to improve performance of the application by efficiently using processor's memory-mapping hardware. THP should reduce the overhead of the Translation Lookaside Buffer. User must login as a sudo user to enable or disable THP settings. It operates mainly in two modes:

- **always:** You can run the following command to set THP to 'always':

```
echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

In this mode, the system kernel tries to assign huge pages to the processes running on the system.

- **madvise:** You can run the following command to set THP to 'madvise':

```
echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
```

In this mode, kernel only assigns huge pages to the individual processes memory areas.

You can use the following command to disable THP:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

It is recommended to use the following THP setting for better performance:

- CNN models - 'always'
- NLP models - 'madvise'

## 7.7 Batch Size

Batch Size is a sensitive factor for the throughput performance of any model. The following formula could be used to calculate the optimal Batch Size:

```
Batch Size = number_of_physical_cores * batch_factor
```

Batch factor may vary from 8-32. The value 32 may provide optimal performance. However, user should verify this empirically.

## 7.8 Memory Allocators

Based on the model, if there is a requirement for a lot of dynamic memory allocations, a memory allocator can be selected from the available allocators which would generate the most optimal performance out of the model. These memory allocators override the system provided dynamic memory allocation routines and use a custom implementation. They also provide the flexibility to override the dynamic memory management specific tunable parameters (for example, logical page size, per thread, or per-cpu cache sizes) and environment variables. The default configuration of these allocators would work well in practice. However, you should verify empirically by trying out what setting works best for a particular model after analyzing the dynamic memory requirements for that model.

Most commonly used allocators are TCMalloc and JEMalloc.

### 7.8.1 TCMalloc

TCMalloc is a memory allocator which is fast, performs uncontended allocation and deallocation for most objects. Objects are cached depending on the mode, either per-thread or per-logical CPU. Most allocations do not need to take locks. So, there is low contention and good scaling for multi-threaded applications. It has flexible use of memory and hence, freed memory can be reused for different object sizes or returned to the operating system. Also, it provides a variety of user-accessible controls that can be tuned based on the memory requirements of the workload.

### 7.8.2 JEMalloc

JEMalloc is a memory allocator that emphasizes fragmentation avoidance and scalable concurrency support. It has a powerful multi-core/multi-thread allocation capability. The more cores the CPU has, the more program threads, the faster JEMalloc allocates. JEMalloc classifies memory allocation

granularity better, leading to less lock contention. It provides various tunable runtime options, such as enabling background threads for unused memory purging, allowing JEMalloc to utilize transparent huge pages for its internal metadata, and so on.

### 7.8.3 Usage

You can install the TCMalloc/JEMalloc dynamic library and use **LD\_PRELOAD** environment variable as follows:

```
Before using TCMalloc:
export LD_PRELOAD=/path/to/TCMallocLib/

Before using JEMalloc:
export LD_PRELOAD=/path/to/JEMallocLib/

Or

Benchmarking command using TCMalloc:
LD_PRELOAD=/path/to/TCMallocLib/ < python benchmarking command>

Benchmarking command using JEMalloc:
LD_PRELOAD=/path/to/JEMallocLib/ < python benchmarking command>
```

To verify if TCMalloc/JEMalloc memory allocator is in use, you can grep for tcmalloc/jemalloc in the output of lsof command:

```
lsof -p <pid_of_benchmarking_commad> | grep <tcmalloc/jemalloc>
```

## Chapter 8 Convolution Algo Logic

Convolution kernels take Input and Filter/Weights as arguments and return Output. The table below describes the expected Layout for each of the convolution algorithms currently supported by TensorFlow-ZenDNN.

**Table 4. Convolution Algo Logic**

zenConvAlgoType	ZENDNN_CONV_ALGO	Input Layout	Filter Layout	Output Layout
GEMM	1	NHWC	HWIO	NHWC
WINOGRAD	2	NHWC	HWIO	NHWC
DIRECT1	3	nChw8c	Ohwi8o	nChw8c
DIRECT2	4	NHWC	Ohwi8o/ Ohwi16o	NHWC

**Note:** In the context of Filter Layouts, HWIO is equivalent to HWCN but with I instead of C representing input channels and O instead of N representing output channels.

## Chapter 9 Limited Precision Support

---

Quantization is an active area of research and a popular compression technique to accelerate neural network performance. Several competitive submissions from MLPerf (<https://mlcommons.org/en/inference-datacenter-10/>) leverage lower precisions to showcase hardware capability.

A few of these quantized neural networks models and TensorFlow protobuf (pb) files are publicly available. On AMD 4<sup>th</sup> Gen EPYC™ platforms, ZenDNN offers options to enable INT8 quantization with AMD's UIF INT8 models. These models can be leveraged using AMD UIF benchmarking scripts.

ZenDNN supports experimental version of BF16 with limited models of AMD UIF model Zoo.

To optimize performance, use the following environment variables:

```
export ZENDNN_CONV_ALGO=4

export ZENDNN_ENABLE_MEMPOOL=1/2

export ZENDNN_TENSOR_POOL_LIMIT=512
```

## Chapter 10 License

---

ZenDNN is licensed under Apache License Version 2.0. Refer to the “LICENSE” file for the full license text and copyright notice.

This distribution includes third party software governed by separate license terms.

### 3-clause BSD license:

- Xbyak (<https://github.com/herumi/xbyak>)
- Googletest (<https://github.com/google/googletest>)
- Instrumentation and Tracing Technology API (<https://github.com/intel/ittapi>)

### Apache License Version 2.0:

- oneDNN (<https://github.com/oneapi-src/oneDNN>)
- Xbyak\_aarch64 ([https://github.com/fujitsu/xbyak\\_aarch64](https://github.com/fujitsu/xbyak_aarch64))
- TensorFlow (<https://github.com/tensorflow/tensorflow>)

### Boost Software License, Version 1.0:

Boost C++ Libraries (<https://www.boost.org/>)

### BSD 2-Clause license:

Caffe (<https://github.com/BVLC/caffe>)

This third-party software, even if included with the distribution of the Advanced Micro Devices software, may be governed by separate license terms, including without limitation, third-party license terms, and open-source software license terms. These separate license terms govern use of the third-party programs as set forth in the THIRD-PARTY-PROGRAMS file.

## Chapter 11 Technical Support

---

Please email [zendnnsupport@amd.com](mailto:zendnnsupport@amd.com) for questions, issues, and feedback on ZenDNN.